NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

PLAN-BASED SIMULATION OF MALICIOUS INTRUDERS ON A COMPUTER SYSTEM

by

Christopher C. Roberts

March 1995

Thesis Advisor: Neil P. Rowe

Approved for public release; distribution is unlimited.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave Blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	March 1995	Master's Thesis

4. TITLE AND SUBTITLE
PLAN-BASED SIMULATION OF MALICIOUS INTRUDERS ON A COMPUTER SYSTEM

5. FUNDING NUMBERS

6. AUTHOR(S)
Roberts, Christopher C.

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Naval Postgraduate School
Monterey, CA 93943-5000

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/ MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSORING/ MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES
The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the United States Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

The problem addressed by this work was to reduce the time taken to train system administrators in detecting computer security problems in system audit logs. The approach taken was to develop a simulator which generates realistic audit logs that illustrate both non-malicious and malicious behavior. These logs can be used to train system administrators. The simulator was written in Prolog and used means-ends analysis to simulate seventeen combinations of general system functions which includes the following: logins, editing, file deletions, file copying, changing file access rights, obtaining superuser privileges, sending mail and logouts. The simulation manipulates virtual system files analogously to what real users do. This creates realistic audit file logs that include a mixture of normal and malicious activity. More impressive is that the entire source program requires only 19.1 kbytes of space, making it small enough to be compatible with a personal computer.

14. SUBJECT TERMS
Artificial Intelligence, Planning, Simulation, Means-Ends Analysis, Intrusion Detection System

15. NUMBER OF PAGES
50

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL

Approved for public release; distribution is unlimited

PLAN-BASED SIMULATION OF MALICIOUS INTRUDERS ON A COMPUTER SYSTEM

Christopher C. Roberts
Lieutenant Commander, United States Naval Reserve
B.S., United States Naval Academy, 1980

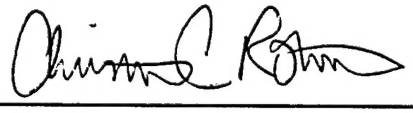
Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

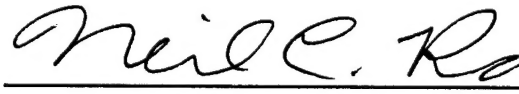
from the

NAVAL POSTGRADUATE SCHOOL

March 1995

Author: 

Christopher C. Roberts

Approved by: 

Neil P. Rowe, Thesis Advisor

Roger Stemp, Second Reader

Ted Lewis, Chairman,
Department of Computer Science

ABSTRACT

The problem addressed by this work was to reduce the time taken to train system administrators in detecting computer security problems in system audit logs. The approach taken was to develop a simulator which generates realistic audit logs that illustrate both non-malicious and malicious behavior. These logs can be used to train system administrators. The simulator was written in Prolog and used means-ends analysis to simulate seventeen combinations of general system functions which includes the following: logins, editing, file deletions, file copying, changing file access rights, obtaining superuser privileges, sending mail and logouts. The simulation manipulates virtual system files analogously to what real users do. This creates realistic audit file logs that include a mixture of normal and malicious activity. More impressive is that the entire source program requires only 19.1 kbytes of space, making it small enough to be compatible with a personal computer.

TABLE OF CONTENTS

I. INTRODUCTION

A. PURPOSE

This thesis simulates normal and malicious ("hacker") computer system usage, and then generates an audit log script. This script could then be used to test automatic Intrusion Detection Systems (IDS) or to tutor system administrators in the detection of threats to the system's security. Normal user actions simulated include file editing, normal log on, becoming a superuser, and sending e-mail messages to warn the system administrator of an attack. The hacker actions to be simulated include file deletion, insertion of a Trojan Horse, file copying, changing file access permissions, and becoming superuser.

B. GENERAL DESCRIPTION

Simulating the normal users and hackers is based on the general activity these two different types of users perform. To ensure effective and realistic simulation, the normal users and hackers are simulated in the Prolog language with general planning techniques and means-ends analysis from METUTOR [Ref. 1]. Prolog's compatibility with means-ends analysis makes it an easy choice to be the programming language used for this simulation.

Hacker simulation is an important developing concept in the field of computer security [Ref. 2]. Although acts of intrusion tend to be sporadic and irregular in nature, a simulation that is generally similar to a hacker can help test intrusion detection. Randomness can be easily included in a means-ends analysis solution to increase realism.

Planning principles are used to simulate the different normal users and hackers and generate specific scripts for each. All of these scripts are then assembled in a single audit file and sorted by time. The normal users and hackers coexist in a virtual world of virtual files. Specification of the actions of normal users and hackers are contained in the "userops" file. The actions are semi-randomly chosen via means-ends analysis. Each action

interacts with a virtual hierarchical file system contained in the "sim_init" file. For instance, at the completion of the file-deletion threat, random files will have been deleted from the virtual system and a one-line audit fact saying so will be written. Randomness is incorporated in some of the operations to simulate the random nature of users. Files are randomly chosen to be edited or removed to simulate the haphazard nature a hacker may exhibit when attacking the system. Hacker goals are simulated to always be different to ensure the greatest degree of realism. Randomness is also used in generating the time duration for each of the different operations.

We also create virtual e-mail messages to the system administrator for simulated users who see something wrong with their files. The odds of a system administrator actually observing a break-in are very slim; the person that is being attacked is actually the system administrator's best "detection device."

C. THESIS CONTENTS

Chapter II provides a brief background on the principles used in the simulation, and compares this simulation with systems that are similar in concept. Chapter III addresses the relation between our simulation and an Intrusion Detection System. Chapter IV describes the simulation in detail, including programming language specifics, input parameters and output parameters. Chapter V includes data on the results ("audit_file" log examples, program running time, size, etc.). Chapter VI describes the major achievements of the simulation, the weaknesses that still exist, and the future research work that could build on this simulation.

II. PLANNING METHODS

This chapter provides background on the planning principles that are used as a basis for the simulation.

A. GENERAL OPERATION

Planning simulation is an integral part of AI-based systems. A common approach to planning involves the action of breaking down a complex problem into smaller, but related, problems. The solutions of the smaller problems are then assembled together to provide the answer to the larger, more complex problem [Ref. 3]. Planning techniques are well matched for dealing with a simulation of the type developed in this thesis. Normal users and hackers are simulated with basic planning principles in mind. Each simple action is designed to cause one primary change to the virtual world. Combining all of the acts together results in a plan [Ref. 4].

It was determined early in the development of AI that simulations that represented the problem-solving knowledge related to the main program would be invaluable. These simulations would allow human experts involved with the expert system to devote more time to the program and related computer configurations [Ref. 5]. A simulation's results are a valuable input to the applicable expert system. To instill a sense of realism to the simulation, uncertainty can be included to vary the outcome.

The simulation uses a searching technique called means-ends analysis. Means-ends analysis is based on analyzing abstractions. Abstractions simplify the simulation, making it easier for the simulation to "reason" [Ref. 6]. Means-ends analysis is a form of backward chaining where the system starts with a set of goals, and then works backwards using rules to see if its possible to obtain the goal. If the rules exist, then the goal can be reached [Ref. 6]. The normal users and hackers are simulated using backward chaining.

Means-ends analysis works on processes that are made up of four main categories: recommended operators, preconditions, addpostconditions and deletepostconditions. Recommended operators are operators that allow the system to advance to the recom-

mended stage by indicating a difference between the current state and the goal state. Recommended operators are in this format [Ref. 6]:

recommended(<difference>,<operator>).

Prior to advancing to a recommended stage, the system has to have the necessary preconditions that are associated with that recommended operator. The operator will fail if the preconditions can not be met. Preconditions are in this format [Ref. 6]:

precondition(<operator>,<list of associated preconditions and predicate expressions that must exist for the operator to succeed>).

The system advances to the goal based on the success of the operators. Means-ends analysis handles changing conditions by using add/deletepostconditions. Addpost-conditions become true after a recommended operator has been completed. Addpost-conditions can be preconditions for another recommended operation. On the other hand, deletepostconditions make the associated operator false and change a fact from true to false. Add/deletepostconditions are in this format [Ref. 6]:

add/deletepostcondition(<opertor>, [<predicate expression1>, <predicate expression2>,...]).

B. GPS

GPS is the first implementation of means-ends analysis, developed by A. Newell and H. Simon in 1963 based on their beliefs that the process behind human reasoning could be modeled by a computer [Ref. 7]. They modeled the human reasoning process based on a comparison process, comparing the initial state of a process to the goal state. These comparisons continue until either the goal state is reached (success) or the process exhausts all of the rules (failure) [Ref. 8]. GPS generates new rules based on the differences that exist between the states. A new rule is used only if it reduces the difference between the states. GPS was also the first successful planning system to use means-ends analysis to reach a goal [Ref. 9]. GPS can backtrack to previously reached goals if the process encounters an unsolvable goal along the way, something important to our program.

C. STRIPS

Like GPS, the Stanford Research Institute Problem Solver (STRIPS) program also exploited preconditions, an add-list, and a delete-list [Ref. 10]. STRIPS's manner in which it proceeds towards a goal is similar to means-ends analysis. But STRIPS's lack of any recommended operator, as in means-ends analysis, can make the process longer as it must search blindly.

III. INTRUSIONS AND THEIR DETECTION

This chapter addresses the relation between our simulation and an Intrusion Detection System (IDS).

A. TOOLS FOR INTRUSION DETECTION

The duties of a system administrator are numerous. They vary from establishing new accounts to ensuring that the system's software remains current. One especially important duty crammed in with all the rest is the responsibility of maintaining system security.

There currently exist numerous automatic tools to assist the system administrator in maintaining system security: log_tcp, Computer Oracle Password and Security System (COPS), Tripwire, to name a few[Ref. 12]. The results of these tools can be directed to a single file so that the system administrator can determine the system status from examining only one source. For instance, one of the functions of COPS is to send a notice to the system administrator when poor passwords are detected. Tripwire detects modifications to the size of specific, sensitive files and sends a report to the system administrator if modifications are detected. The only problem with these reports is that the system administrator must be able to recognize and prioritize a serious threat versus a not-so-serious threat. The system administrator may have a file full of alerts, created by the system security tool, but may fail to understand them. This lapse is easily understandable; without continuous exposure to valid intrusions, threat recognition skills can become rusty.

Automated IDSs exist which utilize an expert system to detect malicious system activity and automatically inform the system administrator of malicious behavior. These systems use two techniques to catch suspicious behavior: statistical inference and rule-based inference[Ref. 13].

Statistical inference is based on comparing observed present behavior to past, established behavior. Audit data is compared with behavioral data that is maintained for

each legitimate user of the system. These two sets of data are each described by a vector, based on intrusion-detection variables. A difference between these two vectors is interpreted as a possible security intrusion[Ref. 13]. The Intrusion-Detection Expert System (IDES) is one type of system that uses statitistical inference.

IDES was developed by a team at the Stanford Research Institute to perform real-time detection of security violations. This system maintains user behavioral profiles based on CPU and memory usage in the form of frequency tables, means and covariances. These statistics are then compared to the audit data with vector differences indicating a possible security intrusion. One problem with this approach is that an intruder's erratic behavior could be similar to a worker who also displays erratic behavior as his "normal behavior." IDES and another IDS, the Multics Intrusion Detection and Alerting System (MIDAS), use rule-based inference techniques to handle such problems [Ref. 13,14].

MIDAS was developed by the National Computer Service Center to monitor a government Multics system. MIDAS uses rule-based inferences to compare actions exhibited in the audit log to expert system rules that are developed based on past intrusions, known system weak points and possible attack scenarios[Ref. 15]. The rules are fixed and are independent of any one particular user or system function. Several examples of behavior covered by MIDAS include the following: password failure on a system account, login failure with an unknown user name, login attempt to an unaccessible account, unusual or invalid commands or command patterns, users logged in simultaneously from two locations, and attempts to modify system files[Ref. 16].

It would be valuable to generate simulations for testing any type of IDS. Then, for instance, IDES can test its effectiveness against variations of the same threat.

B. CURRENT SIMULATION ATTEMPTS

Researchers have recently developed a system prototype which utilizes a tool command language, Tel[Ref. 2]. Tel contains a C-based library package which tests intrusion detection systems by means of running parallel scripts simultaneously. Tel allows

the user to create scripts that resemble actions of an intruder. These scripts are then run as parallel processes so as to resemble multiple attacks occurring simultaneously. The purpose of this prototype is to check the effectiveness of different intrusion detection systems. For the parallel processes to appear realistic, close synchronization is necessary [Ref. 2]. One apparent drawback is that no two runs are predictable unless special synchronization mechanisms are used to augment the system; their system uses such a mechanism.

IV. SIMULATION PROGRAM SPECIFICS

A. ASSUMPTIONS

To develop the simulation, assumptions are made based on typical actions that are made by normal users and hackers. This simulation assumes a clear purpose of each action of normal users and hackers; no "sneaky" actions are simulated. Thus, each action causes an obvious effect to the virtual files found in the data initialization file "sim_init."

One major assumption is that the simulated hacker will eventually be able to log in to the system. In real life, hackers will try repeatedly to log in for hours at a time, quit for the day, and then try again the next day. But a simulated hacker that cannot log in cannot cause simulated damage!

To ensure some degree of randomness to the action of logging in, a probability function is applied to most threat actions. This simulates logins that vary from being able to log in within several attempts, to the dedicated threat that only succeeds in logging in after numerous attempts. But the normal user succeeds in logging in on all attempts.

B. HARDWARE/SOFTWARE REQUIREMENTS

The simulation was developed on a Sun Sparkstation with a UNIX operating system that accesses a Quintus Prolog interpreter. The simulation is written in the Quintus Prolog language, Release 3 [Ref. 17].

C. PROGRAM OPERATIONS

The model consists of four sections: "master," "userops," an input section ("sim_init," "goalsX") and an output section, "audit_file" (see Figure 1 on page 12). The "master" section is the driver section for the simulation. The "userops" section is accessed from the "master" section and contains all of the simulation actions. The "master" section accesses a data initialization file, "sim_init" for input. The "master" section also accesses a "goalsX" file for specific goal completion. The simulation generates a result based on the inputted file facts and the specified parameters in the "master" section, coupled with the

associated operations in the "userops" section. This result is outputted to the "audit_ file." The user of the simulation could then access this "audit_ file" to check on the results.

1. Input Section- "sim_init" File

The data structure of each virtual file in the initialization file "sim_init" is:

*file(**name** of file, parent of owner(i.e. **directory**), **user** of file, **type** of file, **size** of file, read/write/execute **protection** for file, **time** that file was last modified).*

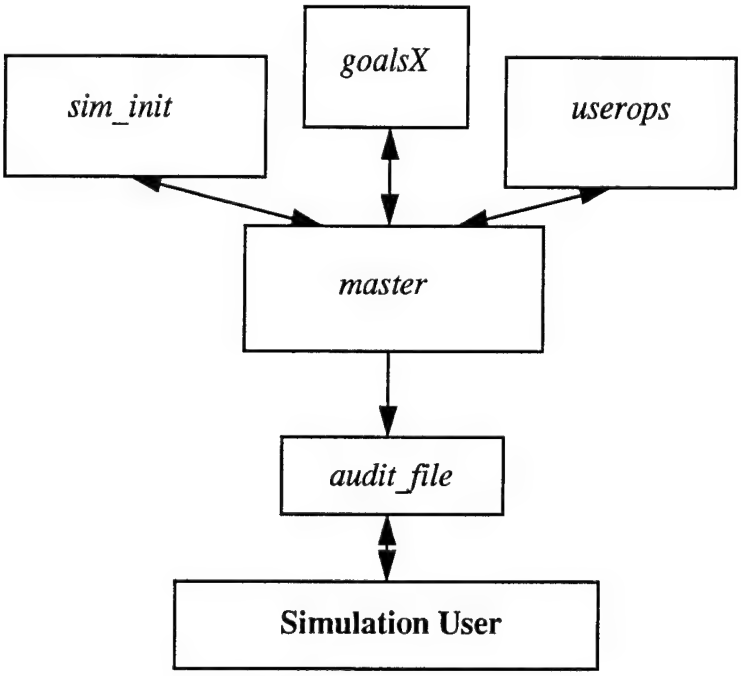


Figure 1: Functional Block Diagram

2. Output Section- "audit_ file" File

The results of each action that occur within the "userops" section are outputted to a virtual "audit_file" file that can be accessed by another system.The data structure of each act in the "audit_file" is:

*audit(**user** of the "sim_init" file, completion **time** of the action on this file, the **directory** that the user is in, **operation** just completed, **status** of operation just completed).*

The last argument to the audit fact, "status," has several meanings. The operation can result in an "ok," "fail," a message to be sent to the system administrator, or a numeric status. A numeric status refers to the changed size of the file that was edited.

3. Goals Section- "goalsX" File

To simulate various users, goal files are created that contain different types of users. Each goal file models the simulated activities of a single user; either a normal user or a malicious user. Specific goal files are selected in the "master" file. The "audit_file" is the resulting mix of running the simulation with all of the simulated users from the different goal files. This allows the flexibility of creating different "audit_files." For instance, if "goals4" had numerous goal facts containing copy file actions, more copy results would exist in the "audit_file" script. Goal facts in each "goalsX" file are arranged like this:

goal([not(logged_in(User)),not(complaint(File,Dir)),modified(ls,bin),
modified(cd,bin)]).

For this particular goal to become true:

not(logged_in(User)): The user must not be logged in.

not(complaint(File,Dir)): The user has no complaints for the system administrator about any files.

modified(ls,bin): File "ls" in "bin" directory has been modified.

modified(cd,bin): File "cd" in "bin" directory has been modified.

4. Driver Section-"master" File

This section loads the libraries, files and another program that provides means-ends analysis for the simulation (METUTOR), starts the simulation and initiates the call to means-ends analysis, and lists the rules that export the output to the "audit_file." In addition, this section runs other routines that affect the overall functioning of the simulation.

a. METUTOR

METUTOR provides the means-ends analysis process for the simulation [Ref. 1]. METUTOR allows each operation to be examined step-by-step as the operations are developed, enabling quick testing of minor refinements.

b. Initiation

This section contains the call to start the simulation, along with the main driving actions, such as the call to the means-ends analysis process, the function that writes to the output file, and the virtual system time initialization function.

To start the simulation, a specific goal file is entered at the prompt; "mygo(1)," for instance, runs the simulation with the goals contained in the file, "goals1." In this section previously altered simulated files are cleared from the buffer each time the simulation is started. For each operation that is run, a virtual system time of completion is generated. Also, a virtual user is selected from one of the "sim_init" file facts. All of the "sim_init" file facts and the goals from the selected goals file are then inputted to the means-ends analysis process.

c. Generate "audit_file" Facts

The section creates "audit_file" facts after a plan for a goal has been formed via the means-ends analysis process. These "audit_file" facts are created based on the operation selected in the initiation section, operation completion time, the virtual user, the virtual directory that the file exists in, and the overall status of this particular operation. Operation names that are generated for the simulation are transformed from within this section into realistic Unix operating names.

d. Modified Files

Professor Rowe developed this section which asserts facts to the file, "tampered_file." This file contains files that have been modified or deleted.

e. Output "audit_file" Facts

Professor Rowe developed this section which takes all of the cached audit facts from all of the goals and writes them out to disc.

f. Eliminate Spurious Output

Professor Rowe developed this section which eliminates spurious output due to rough handling of time, removes complaints about files not yet tampered with, and removes duplicate complaints about the same file from the same user:

5. Specific Operations Section- "userops" File

The "userops" section contains specifications for the different normal and malicious operations. This section contains all of the recommended-operator facts, precondition facts, delete/addpostcondition facts, plus facts and rules that assist the operation of the rules in the "master" section.

To better illustrate these actions, sample facts of several actions are described below:

*recommended([modified(File,Dir)],[file(File,Dir,A,B,C,D,E)],emacs(File,Dir,A, B,C,D,E,DC)):- random(R), DC is integer(300*R)-100.*

In order to modify a file, first a random number "R" is generated. Then, the equation "(300 *R)-100)" is performed; the resulting number "DC" is then converted to an integer. "DC" is now the new size of the modified file.

precondition(cp(File,Dir,File2,Dir2,A,B,C,D,E)[logged_in(User),current_directory(Dir),file(File,Dir,User,B,C,D,E)]).

The preconditions for copying a file are as follows: the user must be logged in and logged in the directory of the file to be copied, and the file that is to be copied must exist in the user's path. The user is randomly selected from the "sim_init" file. The directory of this user is then the target of the copying action.

deletepostcondition(emacs(File,Dir,A,B,C,D,E,F), [file(File,Dir,A,B,C,D,E)]).

Once the protection has been modified, the "file" fact describing the old, unchanged file is no longer true.

addpostcondition(login(User),[current_directory(User),logged_in(User)].

Upon logging in, the user is in the top directory of the login name.

randchange(emacs(File,Dir,A,B,C,D,E,F),[tampered(File,Dir)],complaint(File, Dir),[],0.2).

Ensures that 20% of the time that a file is modified, a complaint mail message is sent to the system administrator if it is thought that the file has been tampered with.

command_result(emacs(File,Dir,A,B,C,D,E,DC), NC):- NC is C+DC,!.

Usually the fifth audit argument is "ok," indicating that the virtual file action completed satisfactorily. But for an "emacs" audit fact, the fifth argument is the new size of the file; determined by adding the old file size to a random number.

average_duration(cp(A,B,C,D,E,F,G,H,I),30).

Copying a file requires 30 seconds on the average.

new_directory(Dir,cd(Dir2),Dir2):-!.
new_directory(Dir,login(User),User):-!.
new_directory(Dir,Op,Dir):-!.

This specifies the new directory a user is in once the command has been completed.

new_account(User,login(User2),User2):-!.new_account(User,O,User).

This is similar to the "new_directory" routine only it specifies the new login name after completing a specific command.

```
acknowledge_tamper(S,G):-
 (member(modified(F,D),G); member(duplicated(F,D,F2,D2),G)),
   \+ tampered(F,D), assertz(tampered_file(F,D)),fail.
acknowledge_tamper(S,G):- member(clobbered(D),G),
   member(file(F,D,_,_,_,_,_),S),
   \+ tampered(F,D), assertz(tampered_file(F,D)),fail.
acknowledge_tamper(S,G):- member(modified(F,D),G),
   member(file(F,D,_,_,_,_,_),S),
   \+ tampered(F,D), assertz(tampered_file(F,D)),fail.
acknowledge_tamper(S,G):- member(accessible(F,D),G),
   member(file(F,D,_,_,_,_,_),S),
```

\+ tampered(F,D), assertz(tampered_file(F,D)), fail.
acknowledge_tamper(S,G).

Specifies conditions under which files can be inferred to have been tampered with.

The "userops" file, with all of the above operator information, is used by the "goalsX" files listing goals for sets of users in simulation runs. The "viewed" goal orders the simulation to perform an "ls" action on the files in the particular directory. The "superused" goal orders the simulation to become a superuser. The "duplicated" goal orders the copying of the file that is in the directory that the user has logged in to. The "modified" goal orders changing the size of the file; sometimes the file size increases, other times it decreases. This change in the size of a file could either be normal use or related to the insertion of a Trojan Horse, virus or other destructive code. The "clobbered" goal orders the deletion of up to six files from the directory that the user is in at the time of logging in. The deletion of a single file is probably related to a normal user action; deletion of several or all files from one directory is more than likely related to malicious activity. The "created" goal orders creating a new file. The "mailed" goal orders sending mail. Upon seeing a malicious action a user invokes this goal to send a mail message to the system administrator, alerting the system administrator to which file is being tampered with. The "logged_in" goal orders logging in to the system under a given user name. The "current_directory" goal orders the user to be in a particular directory. The "accessible" goal orders changing the access rights of a file from any existing protection to no protection at all; this is clearly a malicious action. Some examples of how these goals are achieved are represented by tree structures in Figure 2.

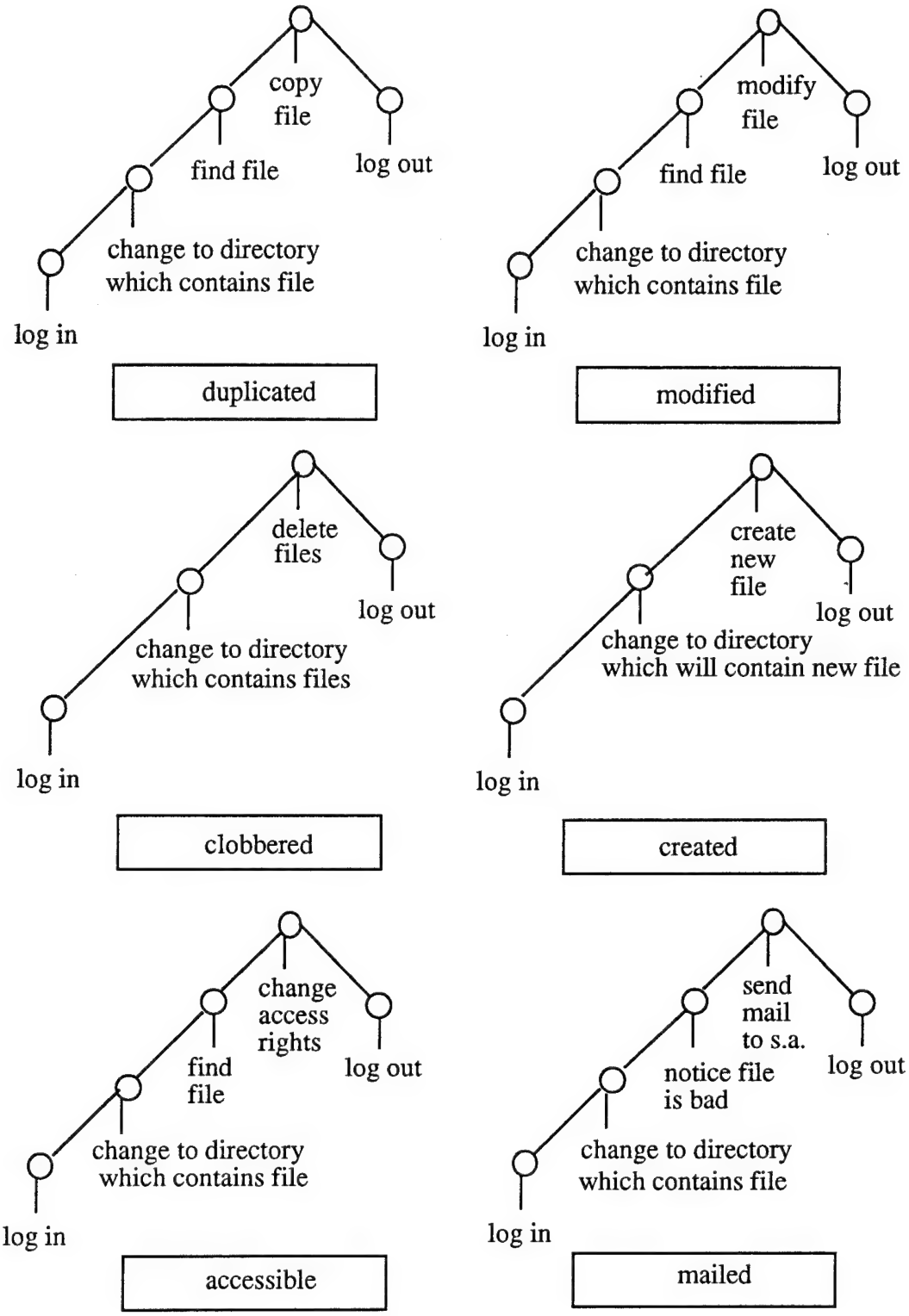


Figure 2: "Userops' Operations

V. RESULTS

Here are sample "audit_file" facts generated for common goals. These appear realistic. Inspection of a complete "audit_file" script (like in Appendix B) reveals that numerous actions can occur concurrently.

MODIFY A FILE

audit(root,2167,none,'login root',ok).
*audit(root,2427,root,'**emacs ls**',2090).* note: "root" uses the emacs editor on file "ls," resulting in a new file size of "2090."
audit(root,2956,root,'emacs cd',5148).
audit(root,2958,root,logout,ok).

MODIFY A FILE AND MAIL MESSAGE TO ROOT

audit(wesson,4368,none,'login wesson',ok).
audit(wesson,4564,wesson,'emacs betterpaper',5416).
*audit(wesson,4648,wesson,'**mail root',bad(betterpaper,wesson)**).* note: "wesson" sends e-mail to the system administrator, "root," to inform "root" that file "betterpaper" in directory "wesson" is suspect.
audit(wesson,4655,wesson,logout,ok).

BECOME SUPERUSER AND CHANGE FILE ACCESS

audit(smith,7603,none,'login smith',ok).
audit(smith,7607,smith,su,ok).
audit(smith,7609,smith,'login jones',ok).
audit(jones,7612,jones,'***chmod 777777 goodpaper***',ok). note: "jones" becomes su and changes the protection on file "goodpaper" to "777777."
audit(jones,7617,jones,logout,ok).

COPY A FILE AND MODIFY A FILE

audit(root,9260,none,'login root',ok).
audit(root,9272,root,'cd bin',ok).
audit(root,9280,bin,'cd etc',ok).
*audit(root,9320,etc,'**cp passwd etc/passwd**',ok).* note: "root" copies file "passwd" in directory "etc."
audit(root,9811,etc,'emacs passwd',977).
audit(root,9813,etc,logout,ok).

BECOME SUPERUSER AND MODIFY A FILE

audit(dow,3917,none,'login dow',ok).

audit(dow,3934,dow,su,ok).
audit(dow,3947,dow,'login smith',ok).
audit(smith,4206,smith,'emacs samples',275).
audit(smith,4212,smith,logout,ok).

BECOME SUPERUSER AND DELETE FILES

audit(dow,9462,none,'login dow',ok).
audit(dow,9467,dow,su,ok).
audit(dow,9481,dow,'login doe',ok).
*audit(doe,9498,doe,'rm *',ok).*
audit(doe,9504,doe,logout,ok).

The statistics for running two different sized goals files shows that a large number of "audit_file" facts can be produced from a small program (total memory: 0.78 mbytes). Running "goals1" took 4.967 seconds, required 19068 bytes of space, ("master," "userops," and "sim_init" require 14323 bytes alone) and generated 163 lines of "audit_file" facts which consisted of 35 users (14 malicious users). Running a smaller file, "goals2", took 1.850 seconds, required 16253 bytes of space, and generated 57 lines of "audit_file" facts which consisted of 15 users (3 malicious users).

VI. CONCLUSIONS

A. MAJOR ACHIEVEMENTS

The simulation generates different sessions of normal and malicious activity in one chronological script. Each action simulated realistically portrays both types of users simultaneously using the virtual system. Our simulation is an excellent input for any system that needs a realistic audit log as an input source. The simulation's structure is flexible enough that it could be used for an expert system that requires numerous chronologically-ordered events as an input. Additionally, it is significant to note that our simulation offers greater portability than an IDS. The size of an IDS usually prohibits any PC application. Our simulation is small and robust enough to be compatible with a personal computer.

B. WEAKNESSES

A weakness is the simplicity of each normal and malicious user action. As there are many different versions of these threats, it would be hard to simulate them all. But any threat-recognition training that a system administrator can receive is more valuable than no training at all.

C. FUTURE RESEARCH

The scope of each action can be broadened and made more complex. As was mentioned in Chapter I, this simulation could be used as an input to an intrusion detection tutor. Additional research could be directed to tailor the simulation to this type of tutor.

Future research could investigate goal manipulation. This could involve tailoring the simulation to provide stimuli for the various automatic IDS mentioned in Chapter III. The simulation could then repeatedly probe an IDS, searching for thin areas of protection, determining which areas are stronger than others, and determining which areas are redundant..

APPENDIX A: PROGRAM CODE

This appendix contains the Prolog code for the "master" file, the "userops" file, the "sim_init" file, and the "goalsX" files.

DRIVER SECTION: "master" file

```
/* Loads metutor30 & operator definition file & initial simulated files */
/* ~rowe/pro/metutor/intdet  */
:- ensure_loaded(library(math)), ensure_loaded('metutor30'),
   ensure_loaded('sim_init'), ensure_loaded(userops),   asserta(debugflag).

/* Argument is an integer indicating which goals set to use */
mygo(N) :- start_state(S), concatenate('goals',N,Goalfile), consult(Goalfile),
   abolish(audit/5), goal(G), mygo(S,G), fail.
mygo(N) :- concatenate('audit',N,Auditfile), write_audit_file(Auditfile).
mygo(XS,G) :- random(R), T is R*10000, round(T,RT),
  nice_bagof(tampered(F2,D2),earlier_tampered_file(F2,D2,RT),TL),
  append(XS,TL,S), abolish(solution/4), abolish(unsolvable/2),
  random_once_means_ends(S,G,Ops,Y), write(Ops), nl, write(Y), nl, nl,
  Ops=[login(U)|_], generate_audit(Ops,RT,U,none), !.
/* Returns a file that has been tampered with before time TT */
earlier_tampered_file(F,D,TT) :- tampered_file(F,D,T), \+ T<TT.
concatenate(S1,S2,S) :- name(S1,A1), name(S2,A2), append(A1,A2,A),
   name(S,A), !.

/* Don't pick directories when you need a random file */
truefile(F,Dir) :- file(F,Dir,_,_,_,_,_), \+ file(_,F,_,_,_,_,_).

/* Create audit facts after a plan for a goal has been found */
generate_audit([],T,U,Dir).
generate_audit([O,O|OL],T,U,Dir) :- !,
   generate_audit_fact(O,T,NT,Dir,U,fail), !, generate_audit([O|OL],NT,U,Dir).
generate_audit([O|OL],T,U,Dir) :- !, command_result(O,CR),
   generate_audit_fact(O,T,NT,Dir,U,CR),
   new_account(U,O,NU), new_directory(Dir,O,NDir), !,
   generate_audit(OL,NT,NU,NDir).

generate_audit_fact(O,T,NT,Dir,U,Status) :- average_duration(O,D), random(X),
   S is X*2*D, NT is T+S,  round(NT,ROT), translate_operator(O,XO),
   note_tampering(O,ROT), assertz(audit(U,ROT,Dir,XO,Status)).

/* Assert facts for files that have been changed or deleted */
```

```
note_tampering(emacs(File,Dir,_,_,_,_,_,_),T) :-
  assertz(tampered_file(File,Dir,T)), !.
note_tampering(newfile(Newname,Dir,_,_,_,_,_),T) :-
  assertz(tampered_file(Newname,Dir,T)), !.
note_tampering(changeaccess(File,Dir,_,_,_,_,_,_),T) :-
  assertz(tampered_file(File,Dir,T)), !.
note_tampering(empty(Dir),T) :- file(File,Dir,_,_,_,_,_),
  assertz(tampered_file(File,Dir,T)), fail.
note_tampering(O,T) :- !.

/* Take all the cached audit facts from all the goals and write       */
/* them out to disk.  Audit file format:                              */
/*  audit(user, ending time of action, directory user is in,          */
/*        operation just completed, status of operation just completed) */

write_audit_file(File) :- tell(File),
  bagof(audit(T,U,DD,A,B),audit(U,T,DD,A,B),L), sort(L,L2),
  fix_erroneous_complaints(L2,L3), member(audit(X,Y,D,Z,W),L3),
  writeq(audit(Y,X,D,Z,W)), write('.'), nl, fail.
write_audit_file(File) :- told.

/* Eliminate spurious output due to rough handling of time:   */
/* remove complaints about files not yet tampered with, and   */
/* remove duplicate complaints about same file from same user */
fix_erroneous_complaints(L2,L3) :-
  no_bt_append(L2a,[audit(T,User,D,'mail root',bad(File,Dir))|L2b],L2),
  (\+ tampered_file_before(File,Dir,T); complaint_before(User,File,Dir,T)), !,
  fix_erroneous_complaints(L2b,L3b), append(L2a,L3b,L3), !.
fix_erroneous_complaints(L2,L3) :-
  no_bt_append(L2a,[audit(T,User,D,'mail root',bad(File,Dir))|L2b],L2), !,
  fix_erroneous_complaints(L2b,L3b),
  append(L2a,[audit(T,User,D,'mail root',bad(File,Dir))|L3b],L3), !.
fix_erroneous_complaints(L,L).
complaint_before(User,File,Dir,T) :-
  audit(User,T2,D,'mail root',bad(File,Dir)), T2<T,
  tampered_file_before(File,Dir,T2), !.
tampered_file_before(File,Dir,T) :- tampered_file(File,Dir,T2), \+ T2>T, !.
no_bt_append(L1,L2,L) :- append(L1,L2,L), !.
```

OPERATIONS SECTION: "userops" file

```
::- ensure_loaded(library(random)).
:- ensure_loaded('sim_init').
```

```
/* Description of the virtual file system:                    */
/* file(filename, directory, user, type, size, protection, timelastchanged) */

recommended([sustatus(User)], su(User)).
recommended([not(complaint(F,D))], mail(root,bad(F,D))).
recommended([viewed(Dir)], ls(Dir)).
recommended([duplicated(File,Dir,File2,Dir2)],[file(File,Dir,A,B,C,D,E)],
   cp(File,Dir,File2,Dir2,A,B,C,D,E)).
recommended([accessible(A,B)],[file(A,B,C,D,E,F,G),
   not(file(A,B,_,_,_,rwxrwx,_))],changeaccess(A,B,C,D,E,F,G,rwxrwx)).
recommended([modified(File,Dir)],[file(File,Dir,A,B,C,D,E)],
   emacs(File,Dir,A,B,C,D,E,DC)) :- random(R), DC is integer(300*R)-100.
recommended([clobbered(Dir)], empty(Dir)).
recommended([created(Newname,Newsize)],[file(File,Dir,User,B,C,D,E)],
   newfile(Newname,Dir,User,'readwrite',Newsize,D,E)).
recommended([mailed(P,M)], mail(P,M)).
recommended([logged_in(User)], login(User)) :- user_name(User).
recommended([current_directory(Dir)], cd(Dir2)) :- random(R),
 ((R>0.6, Dir2=Dir); (\+ R>0.6, all_dirs(Dirs), member(Dir2,Dirs))).
recommended([not(logged_in(User))], logout).

precondition(cp(File,Dir,File2,Dir2,A,B,C,D,E),
  [logged_in(User),current_directory(Dir),file(File,Dir,User,B,C,D,E)]).
precondition(emacs(File,Dir,User,B,C,D,E,F),
  [file(File,Dir,User,B,C,rwxrwx,E)]).
precondition(emacs(File,Dir,User,B,C,D,E,F),
  [logged_in(User)]).
precondition(empty(Dir),[file(File,Dir,User,B,C,rwxrwx,E)]).
precondition(empty(Dir),[file(File,Dir,User,B,C,E,F)],
  [logged_in(User)]).
precondition(newfile(Newname,Dir,User,'readwrite',Newsize,E,F),
  [file(A,Dir,User,C,D,E,F)],[logged_in(User),current_directory(Dir)]).
precondition(cd(Dir),[file(A,Dir,User,C,D,E,F)],
  [not(current_directory(Dir)),logged_in(User)]).
precondition(ls(Dir),[file(A,Dir,User,C,D,E,F)],
  [logged_in(User),current_directory(Dir)]).
precondition(su(User),[not(file(A,Dir,User,D,E,rwxrwx,G))],
  [logged_in(User),not(sustatus(User))]).
precondition(changeaccess(A,B,C,D,E,F,G,NF),[file(A,B,C,D,E,rwxrwx,G)]).
precondition(changeaccess(A,B,C,D,E,F,G,NF),
  [logged_in(C),current_directory(B)]).
precondition(login(User),[not(logged_in(User))]).
```

```
precondition(logout,[logged_in(User)]).
precondition(mail(P,M), [logged_in(User)]).

deletepostcondition(cp(File,Dir,File2,Dir2,A,B,C,D,E),[]).
deletepostcondition(emacs(File,Dir,A,B,C,D,E,F),[tampered(File,Dir)],[]).
deletepostcondition(emacs(File,Dir,A,B,C,D,E,F),[file(File,Dir,A,B,C,D,E)]).
/*  6 files    */
deletepostcondition(empty(B),

[current_directory(B),file(A1,B,C1,D1,E1,F1,G1),file(A2,B,C2,D2,E2,F2,G2),
              file(A3,B,C3,D3,E3,F3,G3),file(A4,B,C4,D4,E4,F4,G4),
              file(A5,B,C5,D5,E5,F5,G5),file(A6,B,C6,D6,E6,F6,G6)],
             [file(A1,B,C1,D1,E1,F1,G1),file(A2,B,C2,D2,E2,F2,G2),
              file(A3,B,C3,D3,E3,F3,G3),file(A4,B,C4,D4,E4,F4,G4),
              file(A5,B,C5,D5,E5,F5,G5),file(A6,B,C6,D6,E6,F6,G6)]).
/*  5 files    */
deletepostcondition(empty(B),
[current_directory(B),file(A1,B,C1,D1,E1,F1,G1),  file(A2,B,C2,D2,E2,F2,G2),
              file(A3,B,C3,D3,E3,F3,G3),  file(A4,B,C4,D4,E4,F4,G4),
              file(A5,B,C5,D5,E5,F5,G5)],[file(A1,B,C1,D1,E1,F1,G1),
              file(A2,B,C2,D2,E2,F2,G2),  file(A3,B,C3,D3,E3,F3,G3),
              file(A4,B,C4,D4,E4,F4,G4),  file(A5,B,C5,D5,E5,F5,G5)]).
/*  4 files    */
deletepostcondition(empty(B),
[current_directory(B),file(A1,B,C1,D1,E1,F1,G1),file(A2,B,C2,D2,E2,F2,G2),
              file(A3,B,C3,D3,E3,F3,G3),file(A4,B,C4,D4,E4,F4,G4)],
             [file(A1,B,C1,D1,E1,F1,G1),file(A2,B,C2,D2,E2,F2,G2),
              file(A3,B,C3,D3,E3,F3,G3),file(A4,B,C4,D4,E4,F4,G4)]).
/*  3 files    *
deletepostcondition(empty(B),
[current_directory(B),file(A1,B,C1,D1,E1,F1,G1),
              file(A2,B,C2,D2,E2,F2,G2),file(A3,B,C3,D3,E3,F3,G3)],
             [file(A1,B,C1,D1,E1,F1,G1),
              file(A2,B,C2,D2,E2,F2,G2),file(A3,B,C3,D3,E3,F3,G3)]).
/*  2 files    */
deletepostcondition(empty(B),
[current_directory(B),file(A1,B,C1,D1,E1,F1,G1),file(A2,B,C2,D2,E2,F2,G2)],
              [file(A1,B,C1,D1,E1,F1,G1),file(A2,B,C2,D2,E2,F2,G2)]).
/*  1 file     */
deletepostcondition(empty(B),
[current_directory(B),file(A1,B,C1,D1,E1,F1,G1)],
              [file(A1,B,C1,D1,E1,F1,G1)]).
deletepostcondition(cd(Dir),[current_directory(Dir2)]).
```

```
deletepostcondition(ls(Dir),[]).
deletepostcondition(su(User),[]).
deletepostcondition(changeaccess(A,B,C,D,E,F,G,P), []).
deletepostcondition(newfile(Newname,Dir,User,'readwrite',Newsize,E,F),[]).
deletepostcondition(login(User),[logged_in(User2)]).
deletepostcondition(logout,[current_directory(Dir),logged_in(User),
            sustatus(User)]).
deletepostcondition(mail(root,bad(F,D)), [complaint(F,D)]).
deletepostcondition(mail(P,M), []).

addpostcondition(ls(Dir), [viewed(Dir)]).
addpostcondition(cp(File,Dir,File2,Dir2,A,B,C,D,E),
  [file(File2,Dir2,A,B,C,D,E),duplicated(File,Dir,File2,Dir2)]).
addpostcondition(emacs(File,Dir,A,B,C,D,E,F),[tampered(File,Dir)],
  [complaint(File,Dir),modified(File,Dir)]).
addpostcondition(emacs(File,Dir,A,B,C,D,E,DC),
  [modified(File,Dir),file(File,Dir,A,B,NC,D,E)]) :-
   var(C); (\+ var(C), NC is C+DC).
addpostcondition(empty(Dir),[clobbered(Dir)]).
addpostcondition(newfile(Newname,Dir,User,'readwrite',Newsize,E,F),
  [created(Newname,Newsize)]).
addpostcondition(cd(Dir), [current_directory(Dir)]).
addpostcondition(su(User), [sustatus(User)]).
addpostcondition(changeaccess(A,B,C,D,E,F,G,NF),
  [accessible(A,B),file(A,B,C,D,E,NF,G)]).
addpostcondition(login(User), [current_directory(User),logged_in(User)]).
addpostcondition(logout, []).
addpostcondition(mail(P,M), [mailed(P,M)]).

randchange(emacs(File,Dir,A,B,C,D,E,F),[tampered(File,Dir)],
  complaint(File,Dir),[],0.2).
randchange(ls(Dir),[],viewed(Dir),[],0.3).
randchange(login(User),[insecure_passwd(User)],[current_directory(Dir),
  logged_in(User)],[],0.6).
randchange(login(User),[],[not(insecure_passwd(User))],[],0.8).

/* Translate from internal names of operators to Unix names for audit file */
translate_operator(cp(File,Dir,File2,Dir2,A,B,C,D,E), S) :-
  name('cp ',AS1), name(File,AS2), name(Dir2,AS3), name(File2,AS4),
  append(AS1,AS2,AS12), append(AS12,[32|AS3],AS123),
  append(AS123,[47|AS4],AS), name(S,AS), !.
translate_operator(emacs(File,Dir,A,B,C,D,E,F), S) :-
  name('emacs ',AS1), name(File,AS2), append(AS1,AS2,AS), name(S,AS), !.
```

```
translate_operator(newfile(Newname,Dir,User,'readwrite',Newsize,E,F), S) :-
  name('emacs ',AS1),name(Newname,AS2),append(AS1,AS2,AS), name(S,AS).
translate_operator(empty(Dir), 'rm *').
translate_operator(cd(File), S) :-
  name('cd ',AS1), name(File,AS2), append(AS1,AS2,AS), name(S,AS), !.
translate_operator(ls(Dir), ls).
translate_operator(su(User), su).
translate_operator(changeaccess(A,B,C,D,E,F,G,P), S) :-
  name('chmod ',AS1), name(777777,AS2), name(A,AS3),
  append(AS1,AS2,AS12), append(AS12,[32|AS3],AS), name(S,AS), !.
translate_operator(login(User), login) :- var(User), !.
translate_operator(login(User), S) :-
  name('login ',AS1), name(User,AS2), append(AS1,AS2,AS), name(S,AS), !.
translate_operator(logout, logout).
translate_operator(mail(P,M), S) :-
  name('mail ',AS1), name(P,AS2), append(AS1,AS2,AS), name(S,AS), !.

/* Last (5th) audit argument is usually "ok", but sometimes not */
command_result(emacs(File,Dir,A,B,C,D,E,DC), NC) :- NC is C+DC, !.
command_result(newfile(Newname,Dir,User,'readwrite',Newsize,E,F), NC) :-
  NC is Newsize, !.
command_result(mail(P,M), M).
command_result(O, ok).

/* Expected duration in seconds for each action */
average_duration(login(User),7).
average_duration(cd(A),10).
average_duration(ls(A),20).
average_duration(su(A),10).
average_duration(empty(A),50).
average_duration(cp(A,B,C,D,E,F,G,H,I),30).
average_duration(emacs(A,B,C,D,E,F,G,H),300).
average_duration(newfile(A,B,C,D,E,F,G),400).
average_duration(changeaccess(A,B,C,D,E,F,G,P),5).
average_duration(logout,4).
average_duration(mail(P,M),100).

/* Routine to calculate new directory you're in after a command */
new_directory(Dir,cd(Dir2),Dir2) :- !.
new_directory(Dir,login(User),User) :- !.
new_directory(Dir,Op,Dir) :- !.

/* Routine to calculate new login name after a command */
```

```
new_account(User,login(User2),User2) :- !.
new_account(User,O,User).

/* Figure which files have been tampered in a state */
acknowledge_tamper(S,G) :-
  (member(modified(F,D),G); member(duplicated(F,D,F2,D2),G)),
   \+ tampered(F,D), assertz(tampered_file(F,D)), fail.
acknowledge_tamper(S,G) :- member(clobbered(D),G),
   member(file(F,D,_,_,_,_,_),S),
   \+ tampered(F,D), assertz(tampered_file(F,D)), fail.
acknowledge_tamper(S,G) :- member(modified(F,D),G),
   member(file(F,D,_,_,_,_,_),S),
   \+ tampered(F,D), assertz(tampered_file(F,D)), fail.
acknowledge_tamper(S,G) :- member(accessible(F,D),G),
   member(file(F,D,_,_,_,_,_),S),
   \+ tampered(F,D), assertz(tampered_file(F,D)), fail.
acknowledge_tamper(S,G).

start_state(P1):-
   bagof(file(A,B,C,D,E,F,G),A^B^C^D^E^F^G^file(A,B,C,D,E,F,G),P1).
random_user(User) :- setof(U,A^B^C^D^E^F^file(A,B,U,C,D,E,F),UL),
   randitem(UL,User).
user_name(User) :- setof(U,A^B^C^D^E^F^file(A,B,U,C,D,E,F),UL),
   member(User,UL).
all_dirs(Dirs) :- setof(B,A^C^D^E^F^G^file(A,B,C,D,E,F,G),Dirs).
```

INITIALIZATION SECTION: "sim_init" file

```
/* Description of a simulated file system: */
/* filename, directory, user, type, size, protection, timelastchanged */

:- dynamic newname/1.

file(bin,root,root,directory,100,'r--r--',10).
file(ls,bin,root,executable,2000,'r--r--',20).
file(cd,bin,root,executable,5000,'r--r--',30).
file(passwd,etc,root,readwrite,1000,'r--r--',40).
file(doe,users,doe,readwrite,100,'r--r--',100).
file(bigpaper,doe,doe,readwrite,30000,'r--r--',500).
file(samples,smith,smith,readwrite,100,'r--r--',100).
file(shortpaper,smith,smith,readwrite,5400,'r--r--',500).
file(goodpaper,jones,jones,readwrite,5400,'r--r--',500).
file(betterpaper,wesson,wesson,readwrite,5400,'r--r--',500).
```

```
file(bestpaper,dow,dow,readwrite,5400,'r--r--',500).

/* new names for newly created files  */
newname(tmp127).
newname(tmp128).
newname(tmp129).
newname(tmp130).
newname(tmp131).
newname(tmp132).
newname(tmp133).

/* user names that have insecure passwords  */
insecure_password(smith).
insecure_password(doe).
```

GOALS SECTION: "goals1" file

```
/* Simulated-user goals */
:- dynamic newname/1.

goal([not(logged_in(User)),not(complaint(F,D)),modified(ls,bin),
    modified(cd,bin)]).
goal([not(logged_in(User)),modified(bigpaper,doe),viewed(doe)]).
goal([not(logged_in(User)),sustatus(dow),modified(samples,smith)]).
goal([not(logged_in(User)),accessible(bigpaper,doe),modified(bigpaper,doe)]).
goal([not(logged_in(User)),duplicated(passwd,etc,F,D),modified(passwd,etc)]).
goal([not(logged_in(User)),viewed(root)]).
goal([not(logged_in(User)),accessible(doe,users),modified(doe,users)]).
goal([not(logged_in(User)),viewed(smith),modified(shortpaper,smith)]).
goal([not(logged_in(User)),created(Newname,Newsize)]) :-
    bagof(A,newname(A),NNL),randitem(NNL,Newname),
    retract(newname(Newname)),random(R), Newsize is integer(1400*R)-100.
goal([not(logged_in(User)),created(Newname,Newsize),
    duplicated(bin,root,F,D)]) :-
    bagof(A,newname(A),NNL),randitem(NNL,Newname),
    retract(newname(Newname)),random(R), Newsize is integer(1400*R)-100.
goal([not(logged_in(User)),duplicated(samples,root,F,D)]).
goal([not(logged_in(User)),created(Newname,Newsize),
    modified(shortpaper,smith)]) :-
    bagof(A,newname(A),NNL),randitem(NNL,Newname),
    retract(newname(Newname)),random(R), Newsize is integer(1400*R)+100.
goal([not(logged_in(User)),created(Newname,Newsize),
    duplicated(bigpaper,doe,F,D),modified(bigpaper,doe)]) :-
```

```
    bagof(A,newname(A),NNL),randitem(NNL,Newname),
    retract(newname(Newname)),random(R), Newsize is integer(1400*R)+100.
goal([not(logged_in(User)),sustatus(dow), modified(betterpaper,wesson)]).
goal([not(logged_in(User)),not(complaint(F,D)),viewed(root)]).
goal([not(logged_in(User)),modified(goodpaper,jones),viewed(jones)]).
goal([not(logged_in(User)),not(complaint(F,D)),modified(betterpaper,wesson)]).
goal([not(logged_in(User)),accessible(bestpaper,dow),modified(bestpaper,dow)]).
goal([not(logged_in(User)),sustatus(smith),accessible(goodpaper,jones)]).
goal([not(logged_in(User)),modified(ls,bin),modified(cd,bin)]).
goal([not(logged_in(User)),modified(bigpaper,doe),viewed(doe)]).
goal([not(logged_in(User)),not(complaint(F,D)),modified(passwd,etc)]).
goal([not(logged_in(User)),not(complaint(F,D)),accessible(bigpaper,doe),
    modified(bigpaper,doe)]).
goal([not(logged_in(User)),not(complaint(F,D)),duplicated(passwd,etc,F,D),
    modified(passwd,etc),accessible(passwd,etc)]).
goal([not(logged_in(User)),not(complaint(F,D)),accessible(doe,users),
    modified(doe,users)]).
goal([not(logged_in(User)),viewed(smith),modified(shortpaper,smith)]).
goal([not(logged_in(User)),created(Newname,Newsize)]) :-
    bagof(A,newname(A),NNL),randitem(NNL,Newname),
    retract(newname(Newname)),random(R), Newsize is integer(1400*R)+100.
goal([not(logged_in(User)),created(Newname,Newsize)]) :-
    bagof(A,newname(A),NNL),randitem(NNL,Newname),
    retract(newname(Newname)),random(R), Newsize is integer(1400*R)+100.
goal([not(logged_in(User)),duplicated(samples,root,F,D)]).
goal([not(logged_in(User)),created(Newname,Newsize),
    modified(shortpaper,smith)]) :-
    bagof(A,newname(A),NNL),randitem(NNL,Newname),
    retract(newname(Newname)),random(R), Newsize is integer(1400*R)+100.
goal([not(logged_in(User)),viewed(smith),modified(shortpaper,smith)]).
goal([not(logged_in(User)),created(Newname,Newsize)],
   [duplicated(bigpaper,doe,F,D),modified(bigpaper,doe)]) :-
    bagof(A,newname(A),NNL),randitem(NNL,Newname),
    retract(newname(Newname)),random(R), Newsize is integer(1400*R)+100.
goal([not(logged_in(User)),sustatus(dow),clobbered(doe)]).
goal([not(logged_in(User)),viewed(wesson)]).
goal([not(logged_in(User)),not(complaint(F,D)),viewed(root)]).
goal([not(logged_in(User)),
    modified(goodpaper,jones),viewed(jones)]).
goal([not(logged_in(User)),not(complaint(F,D)),modified(betterpaper,wesson)]).
goal([not(logged_in(User)),not(complaint(F,D)),accessible(bestpaper,dow),
    modified(bestpaper,dow)]).
```

```
/* new names for new files */

newname(tmp127).
newname(tmp128).
newname(tmp129).
newname(tmp130).
newname(tmp131).
newname(tmp132).
newname(tmp133).
```

GOALS SECTION: "goals2" file

```
/* Simulated-user goals */
:- dynamic newname/1.

goal([not(logged_in(User)),not(complaint(F,D)),modified(bigpaper,doe),
    viewed(doe)]).
goal([not(logged_in(User)),created(Newname,Newsize)]) :-
    bagof(A,newname(A),NNL),randitem(NNL,Newname),
    retract(newname(Newname)),random(R), Newsize is integer(1400*R)+100.
goal([not(logged_in(User)),modified(bin,root)]).
goal([not(logged_in(User)),modified(samples,smith)]).
goal([not(logged_in(User)),viewed(etc),duplicated(passwd,etc,F,D)]).
goal([not(logged_in(User)),modified(goodpaper,jones)]).
goal([not(logged_in(User)),not(complaint(F,D)),viewed(dow)]).
goal([not(logged_in(User)),sustatus(dow),modified(samples,smith)]).
goal([not(logged_in(User)),duplicated(bestpaper,dow,F,D)]).
goal([not(logged_in(User)),modified(bigpaper,doe),accessible(bigpaper,doe)]).
goal([not(logged_in(User)),not(complaint(F,D)),viewed(bin)]).
goal([not(logged_in(User)),created(Newname,Newsize)]) :-
    bagof(A,newname(A),NNL),randitem(NNL,Newname),
    retract(newname(Newname)),random(R), Newsize is integer(1400*R)+100.
goal([not(logged_in(User)),accessible(betterpaper,wesson)]).
goal([not(logged_in(User)),viewed(jones),mailed(goodpaper,jones)]).
goal([not(logged_in(User)),sustatus(jones),clobbered(wesson)]).
/* new names for new files */
newname(tmp127).
newname(tmp128).
newname(tmp129).
newname(tmp130).
newname(tmp131).
newname(tmp132).
newname(tmp133).
```

APPENDIX B: PROGRAM RESULTS

This appendix contains the results of the simulation's execution. These results are found in the "auditX" file.

Audit file format:
audit(**user**, ending **time** of action, **directory** user is in,
operation just completed, **status** of operation just completed or **size** of new file)

RESULTS: "audit1" file (the result of running "goals1"

audit(jones,190,none,'login jones',ok).
audit(jones,242,jones,'emacs goodpaper',5550).
audit(jones,268,jones,ls,ok).
audit(jones,270,jones,logout,ok).
audit(smith,298,none,'login smith',ok).
audit(smith,320,smith,'emacs shortpaper',5525).
audit(smith,329,smith,'login dow',ok).
audit(dow,661,dow,'emacs tmp127',1142).
audit(dow,662,dow,logout,ok).
audit(wesson,884,none,'login wesson',ok).
audit(wesson,1386,wesson,'emacs betterpaper',5508).
audit(wesson,1392,wesson,logout,ok).
audit(doe,1604,none,'login doe',ok).
audit(root,1684,none,'login root',ok).
audit(root,1695,none,'login root',ok).
audit(root,1707,root,ls,ok).
audit(root,1714,root,logout,ok).
audit(doe,2160,none,'login doe',ok).
audit(doe,2163,doe,'emacs bigpaper',30037).
audit(doe,2165,doe,'chmod 777777 bigpaper',ok).
audit(root,2167,none,'login root',ok).
audit(doe,2168,doe,ls,ok).
audit(doe,2173,doe,logout,ok).
audit(root,2193,root,'emacs ls',1937).
audit(doe,2218,doe,'emacs bigpaper',29979).
audit(doe,2221,doe,logout,ok).
audit(root,2342,root,'emacs cd',5006).
audit(root,2346,root,logout,ok).
audit(root,2427,root,'emacs ls',2090).
audit(root,2956,root,'emacs cd',5148).
audit(root,2958,root,logout,ok).
audit(dow,3612,none,'login dow',ok).

```
audit(dow,3620,dow,'chmod 777777 bestpaper',ok).
audit(root,3830,none,'login root',ok).
audit(root,3832,root,'cd etc',ok).
audit(root,3883,etc,'cp passwd etc/passwd',ok).
audit(root,3891,etc,'chmod 777777 passwd',ok).
audit(dow,3917,none,'login dow',ok).
audit(dow,3934,dow,su,ok).
audit(dow,3947,dow,'login smith',ok).
audit(dow,4147,dow,'emacs bestpaper',5324).
audit(dow,4152,dow,logout,ok).
audit(dow,4153,none,'login dow',ok).
audit(dow,4157,dow,su,ok).
audit(dow,4160,dow,'login wesson',ok).
audit(smith,4206,smith,'emacs samples',275).
audit(smith,4212,smith,logout,ok).
audit(wesson,4368,none,'login wesson',ok).
audit(root,4392,etc,'emacs passwd',1087).
audit(root,4430,etc,'mail root',bad(passwd,etc)).
audit(root,4432,etc,logout,ok).
audit(wesson,4564,wesson,'emacs betterpaper',5416).
audit(wesson,4619,wesson,'emacs betterpaper',5497).
audit(wesson,4624,wesson,logout,ok).
audit(wesson,4648,wesson,'mail root',bad(betterpaper,wesson)).
audit(doe,4650,none,'login doe',ok).
audit(doe,4655,doe,'login root',ok).
audit(wesson,4655,wesson,logout,ok).
audit(root,4662,root,'cd bin',ok).
audit(root,4664,bin,'login doe',ok).
audit(doe,4673,doe,'login root',ok).
audit(root,4674,root,'cd etc',ok).
audit(root,4686,etc,'login doe',ok).
audit(doe,4692,doe,'cd users',ok).
audit(doe,4701,users,'chmod 777777 doe',ok).
audit(doe,4946,users,'emacs doe',102).
audit(doe,4948,users,logout,ok).
audit(root,4997,none,'login root',ok).
audit(root,5668,root,'emacs tmp131',1224).
audit(root,5671,root,logout,ok).
audit(doe,6318,none,'login doe',ok).
audit(doe,6342,doe,'cp bigpaper doe/bigpaper',ok).
audit(dow,6374,none,'login dow',ok).
audit(dow,6375,dow,'chmod 777777 bestpaper',ok).
audit(doe,6741,doe,'emacs bigpaper',30024).
```

audit(doe,6742,doe,'login dow',ok).
audit(dow,6765,dow,'emacs tmp132',135).
audit(dow,6770,dow,logout,ok).
audit(dow,6771,dow,'emacs bestpaper',5318).
audit(dow,6773,dow,logout,ok).
audit(smith,7437,none,'login smith',ok).
audit(root,7451,none,'login root',ok).
audit(doe,7477,none,'login doe',ok).
audit(doe,7484,doe,'login root',ok).
audit(root,7491,root,ls,fail).
audit(root,7501,root,'cd bin',ok).
audit(root,7503,bin,'login doe',ok).
audit(root,7504,root,ls,ok).
audit(root,7505,root,logout,ok).
audit(doe,7515,doe,'login root',ok).
audit(root,7516,root,'cd etc',ok).
audit(root,7522,etc,'login doe',ok).
audit(doe,7535,doe,'cd users',ok).
audit(doe,7540,users,'chmod 777777 doe',ok).
audit(doe,7567,users,'emacs doe',136).
audit(doe,7569,users,logout,ok).
audit(smith,7603,none,'login smith',ok).
audit(smith,7607,smith,su,ok).
audit(smith,7609,smith,'login jones',ok).
audit(jones,7612,jones,'chmod 777777 goodpaper',ok).
audit(jones,7617,jones,logout,ok).
audit(root,7786,none,'login root',ok).
audit(root,7828,root,'cp bin root/bin',ok).
audit(root,7836,root,'login dow',ok).
audit(smith,7921,smith,'emacs shortpaper',5498).
audit(jones,7931,none,'login jones',ok).
audit(smith,7934,smith,'login dow',ok).
audit(root,8108,none,'login root',ok).
audit(smith,8172,none,'login smith',ok).
audit(jones,8223,jones,'emacs goodpaper',5445).
audit(jones,8252,jones,ls,ok).
audit(jones,8256,jones,logout,ok).
audit(dow,8316,dow,'emacs tmp128',985).
audit(dow,8321,dow,logout,ok).
audit(root,8346,root,'emacs tmp133',255).
audit(root,8349,root,logout,ok).
audit(smith,8540,smith,'emacs shortpaper',5368).
audit(smith,8570,smith,ls,ok).

```
audit(smith,8571,smith,logout,ok).
audit(dow,8598,dow,'emacs tmp129',599).
audit(dow,8604,dow,logout,ok).
audit(root,8654,none,'login root',ok).
audit(root,8754,root,'emacs passwd',1147).
audit(root,8759,root,logout,ok).
audit(doe,8861,none,'login doe',ok).
audit(doe,8982,doe,'emacs bigpaper',29909).
audit(doe,8990,doe,ls,ok).
audit(doe,8991,doe,logout,ok).
audit(smith,8994,none,'login smith',ok).
audit(smith,9068,smith,'emacs shortpaper',5418).
audit(smith,9093,smith,ls,ok).
audit(smith,9097,smith,logout,ok).
audit(doe,9193,none,'login doe',ok).
audit(doe,9196,doe,'chmod 777777 bigpaper',ok).
audit(root,9260,none,'login root',ok).
audit(root,9272,root,'cd bin',ok).
audit(root,9280,bin,'cd etc',ok).
audit(root,9320,etc,'cp passwd etc/passwd',ok).
audit(doe,9416,doe,'emacs bigpaper',30186).
audit(doe,9424,doe,logout,ok).
audit(dow,9462,none,'login dow',ok).
audit(dow,9467,dow,su,ok).
audit(dow,9481,dow,'login doe',ok).
audit(doe,9498,doe,'rm *',ok).
audit(doe,9504,doe,logout,ok).
audit(wesson,9645,none,'login wesson',ok).
audit(wesson,9656,wesson,ls,fail).
audit(wesson,9694,wesson,ls,fail).
audit(root,9707,none,'login root',ok).
audit(wesson,9715,wesson,ls,ok).
audit(wesson,9722,wesson,logout,ok).
audit(root,9811,etc,'emacs passwd',977).
audit(root,9813,etc,logout,ok).
audit(root,9851,none,'login root',ok).
audit(smith,9854,none,'login smith',ok).
audit(root,9874,root,ls,ok).
audit(root,9876,root,logout,ok).
audit(root,10149,root,'emacs tmp130',1023).
audit(root,10153,root,logout,ok).
audit(smith,10449,smith,'emacs shortpaper',5535).
audit(smith,10475,smith,ls,fail).
```

audit(smith,10514,smith,ls,ok).
audit(smith,10519,smith,logout,ok).

RESULTS: "audit2" file (the results of running "goals2"

audit(smith,58,none,'login smith',ok).
audit(smith,253,smith,'emacs samples',269).
audit(smith,257,smith,logout,ok).
audit(doe,922,none,'login doe',ok).
audit(doe,931,doe,'chmod 777777 bigpaper',ok).
audit(doe,1352,doe,'emacs bigpaper',29943).
audit(doe,1358,doe,logout,ok).
audit(root,1598,none,'login root',ok).
audit(root,1727,root,'emacs tmp133',515).
audit(root,1733,root,logout,ok).
audit(doe,2166,none,'login doe',ok).
audit(doe,2562,doe,'emacs bigpaper',30090).
audit(doe,2572,doe,ls,ok).
audit(doe,2576,doe,logout,ok).
audit(dow,3263,none,'login dow',ok).
audit(dow,3310,dow,'cp bestpaper dow/bestpaper',ok).
audit(dow,3317,dow,logout,ok).
audit(jones,3349,none,'login jones',ok).
audit(jones,3363,jones,su,ok).
audit(jones,3364,jones,'login wesson',ok).
audit(wesson,3440,wesson,'rm *',ok).
audit(wesson,3441,wesson,logout,ok).
audit(root,3611,none,'login root',ok).
audit(root,3823,none,'login root',ok).
audit(root,3841,root,'cd bin',ok).
audit(root,3858,bin,'cd etc',ok).
audit(root,3864,root,'emacs bin',5).
audit(root,3868,root,logout,ok).
audit(root,3874,etc,'cp passwd etc/passwd',ok).
audit(root,3896,etc,ls,ok).
audit(root,3898,etc,logout,ok).
audit(wesson,7786,none,'login wesson',ok).
audit(wesson,7791,wesson,'chmod 777777 betterpaper',ok).
audit(wesson,7793,wesson,logout,ok).
audit(jones,8295,none,'login jones',ok).
audit(dow,8347,none,'login dow',ok).
audit(dow,8386,dow,ls,ok).
audit(dow,8394,dow,logout,ok).

```
audit(root,8394,none,'login root',ok).
audit(root,8400,root,'cd bin',ok).
audit(root,8436,bin,ls,fail).
audit(root,8453,bin,ls,ok).
audit(root,8455,bin,logout,ok).
audit(root,8594,none,'login root',ok).
audit(jones,8696,jones,'emacs goodpaper',5511).
audit(jones,8702,jones,logout,ok).
audit(root,8759,root,'emacs tmp132',1199).
audit(root,8764,root,logout,ok).
audit(dow,8839,none,'login dow',ok).
audit(dow,8841,dow,su,ok).
audit(dow,8846,dow,'login smith',ok).
audit(jones,9246,none,'login jones',ok).
audit(jones,9252,jones,ls,fail).
audit(jones,9267,jones,ls,ok).
audit(jones,9370,jones,'mail goodpaper',jones).
audit(jones,9375,jones,logout,ok).
audit(smith,9391,smith,'emacs samples',182).
audit(smith,9395,smith,logout,ok).
```

LIST OF REFERENCES

1 Rowe, Neil C., and Galvin, Thomas P., "An Authoring System for Intelligent Tutors for Procedural Skills," Department of Computer Science, Naval Postgraduate School, Monterey, CA.

2 Puketza, Nicholas, Zhang, Biswanath, Mukherjee Kui, and Olsson, Ronald A., "Testing Intrusion Detection Systems: Design Methodologies and Results from an Early Prototype," Department of Computer Science, University of California, Davis, CA.

3 Feigenbaum, E.A., and Feldman, J., "Computers and Thought," *McGraw-Hill*, New York, 1963.

4 Nilsson, Nils J., "Readings in Planning, " *Morgan Kaufmann Publishers, Inc.*, 1990.

5 Kulikowski, C.A., Huber, R.M., and Ferrate, G.A., "Artificial Intelligence, Expert Systems and Languages in Modeling and Simulation," *North-Holland*, New York, 1988.

6 Rowe, Neil C., "Artificial Intelligence Through Prolog," *Prentice-Hall*, Englewood Cliffs, NJ, 1988.

7 Newell, Allen, and Simon, H.A., "GPS: A Program that Simulates Human Thought," *Lerende Automaten*, R. Oldenbourg, KG, 1961.

8 Forsyth, Richard, "Expert Systems," *Chapman and Hall Computing*, New York, 1984.

9 Ernst, George W., and Newell, Allen, "GPS: A Case Study in Generality and Problem Solving," *Academic Press*, New York, 1969

10 Tate, Austin, Hendler, James, and Drummond, Mark, "A Review of AI Planning Techniques," AI Research Branch, NASA Ames Research Center, Mountain View, CA.

11 Georgeff, Michael P., "Planning," Annual Review of Computer Science, Volume 2, *Annual Reviews, Inc.*, 1987.

12 Rich, Lyford D., "Unix Security: A Penetration Analysis of Navy Computer Systems," Master's Thesis, Naval Postgraduate School, Monterey, CA,1992.

13 Lunt, Teresa F., Jagannathan, R., Lee, Rosana, Whitehurst, Alan, and Listgarten, Sherry, "Knowledge-Based Intrusion Detection," *Proceedings of the 1989 AI Systems in Government Conference*, March, 1989.

14 Denning, D. E., and Neumann, P. G., "Requirements and Model for IDES- A Real-Time Intrusion Detection System," Computer Science Laboratory, SRI International, 1985.

15 Lunt, Teresa F., "Real-Time Intrusion Detection," *Proceedings of COMPCON Spring'89*, San Francisco, CA, February, 1989.

16 Lunt, Teresa F., "Automated Audit Trail Analysis," *Proceedings of the 11th National Computer Security Conference*," Baltimore, MD, October, 1988.

17 Quintus Prolog, *Release 3*, Quintus Corporation, Palo Alto, CA.

INITIAL DISTRIBUTION LIST

1 Defense Technical Information Center 2
Cameron Station
Alexandria, VA 22304-6145

2 Dudley Knox Library 2
Code 52
Naval Postgraduate School
Monterey, CA 93943-5101

3 Chairman, Code CS 2
Computer Science Department
Naval Postgraduate School
Monterey, CA 93943

4 Dr. Neil Rowe, Code CS/Rp 2
Computer Science Department
Naval Postgraduate School
Monterey, CA 93943

5 Roger Stemp, Code CS/Sp 2
Computer Science Department
Naval Postgraduate School
Monterey, CA 93943

6 LCDR Christopher C. Roberts 2
1023 Cassia Way
Sunnyvale, CA 94086